

A Systematic Approach Towards Missing Lab Data in Electronic Health Records: A Case Study in Non-Small Cell Lung Cancer and Multiple Myeloma

Arjun Sondhi,^{1*} Janick Weerpals,^{2*} Prakirthi Yerram,¹ Chengsheng Jiang,¹ Michael Taylor,³

Meghna Samant,¹ Sarah Cherng¹

¹ Flatiron Health, Inc., New York, NY, USA

² Hoffmann-La Roche, Basel, Switzerland

³ Genentech, San Francisco, CA, USA

* These authors contributed equally to this work.

Corresponding author:

Arjun Sondhi

Flatiron Health

233 Spring St, 5th Fl

New York, NY 10013

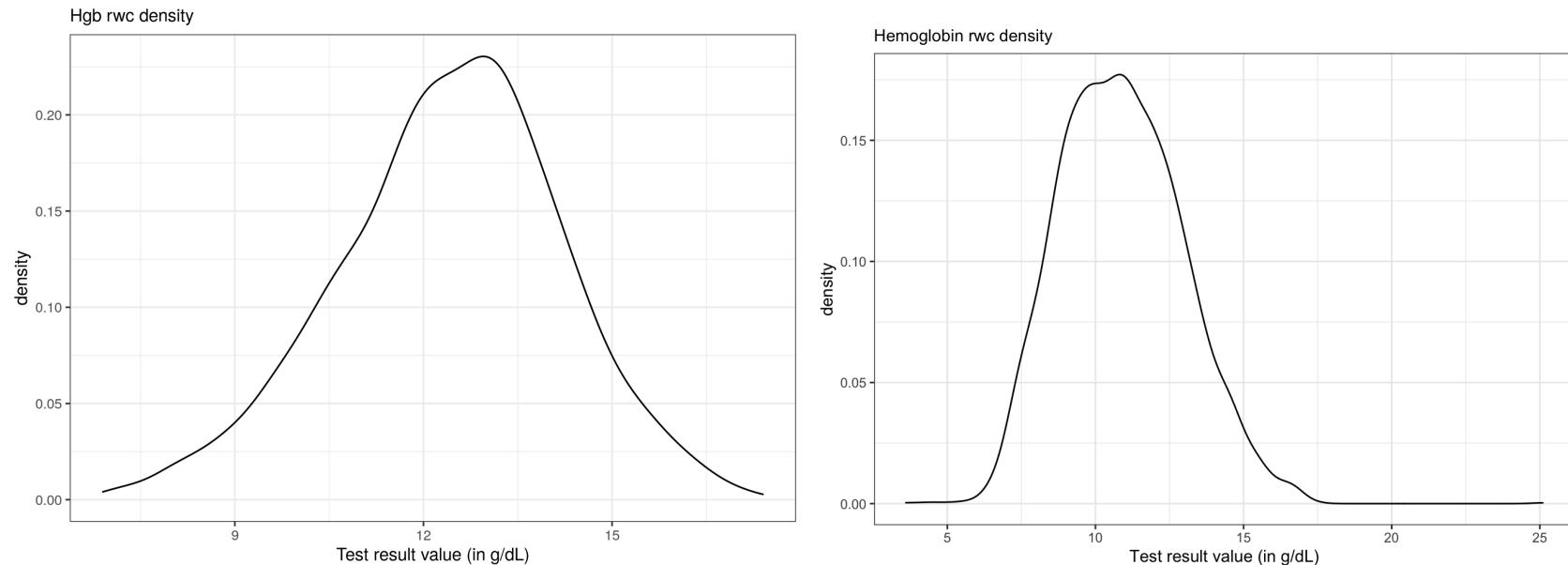
Email: arjun.sondhi@flatiron.com

Supplement

Supplementary Figure 1. Baseline distribution of labs in advanced non-small cell lung cancer (left) and multiple myeloma (right) real-world cohorts.	3
Supplementary Figure 2. Comparison and balance of patient characteristics with and without observed labs in advanced non-small cell lung cancer (top) and multiple myeloma (bottom) real-world cohorts.	4
Supplementary Figure 3. Illustration of hyperparameter tuning for AUC diagnostics classifier model in advanced non-small cell lung cancer (top) and multiple myeloma (bottom) real-world cohorts.	6
Multiple myeloma (MM)	7
Supplementary Figure 4. Variable importance in predicting missingness in advanced non-small cell lung cancer (top) and multiple myeloma (bottom) real-world cohorts.	8
SUPPLEMENTARY TABLES	10
Supplementary Table 1. Details on applied real-world cohort case studies.	11
Supplementary Table 2. Results of Hotelling's test	11
SUPPLEMENTARY SIMULATION CODE	12

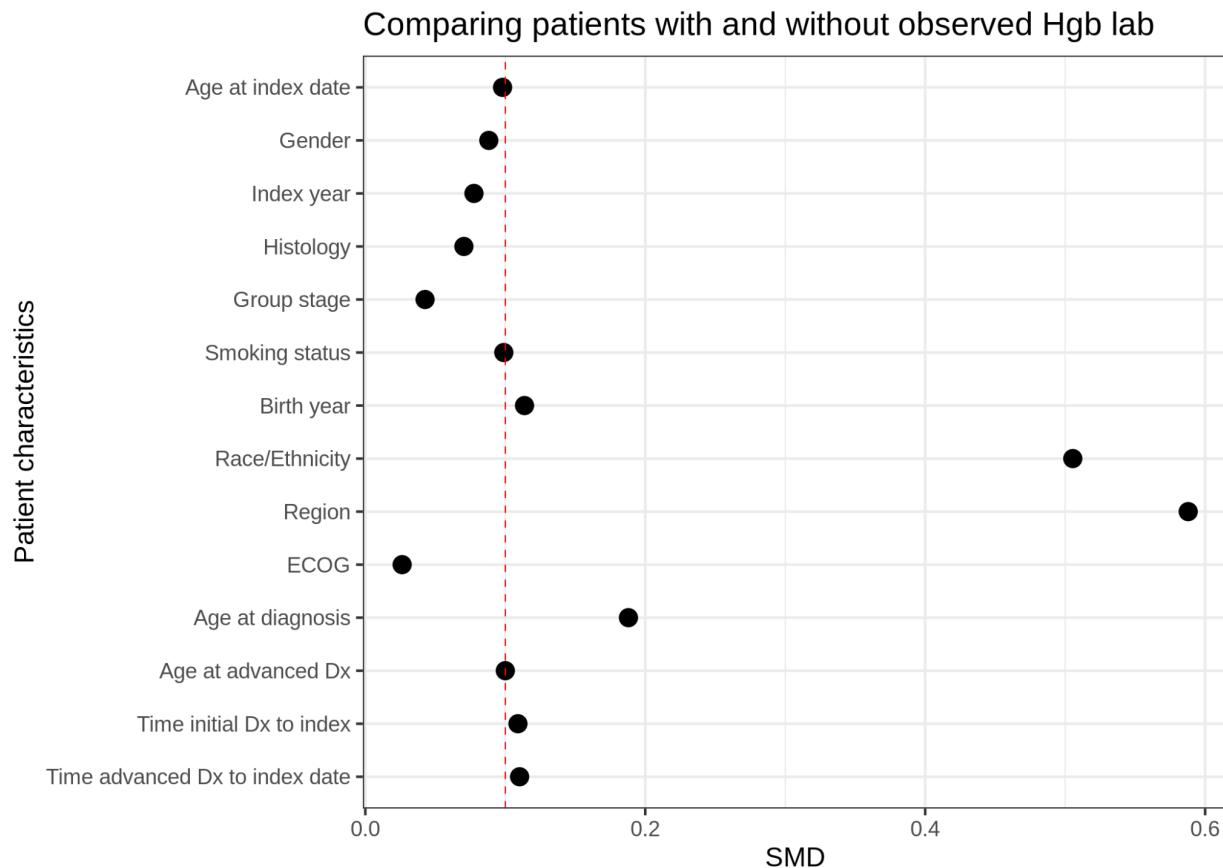
SUPPLEMENTARY FIGURES

Supplementary Figure 1. Baseline distribution of labs in advanced non-small cell lung cancer (left) and multiple myeloma (right) real-world cohorts.



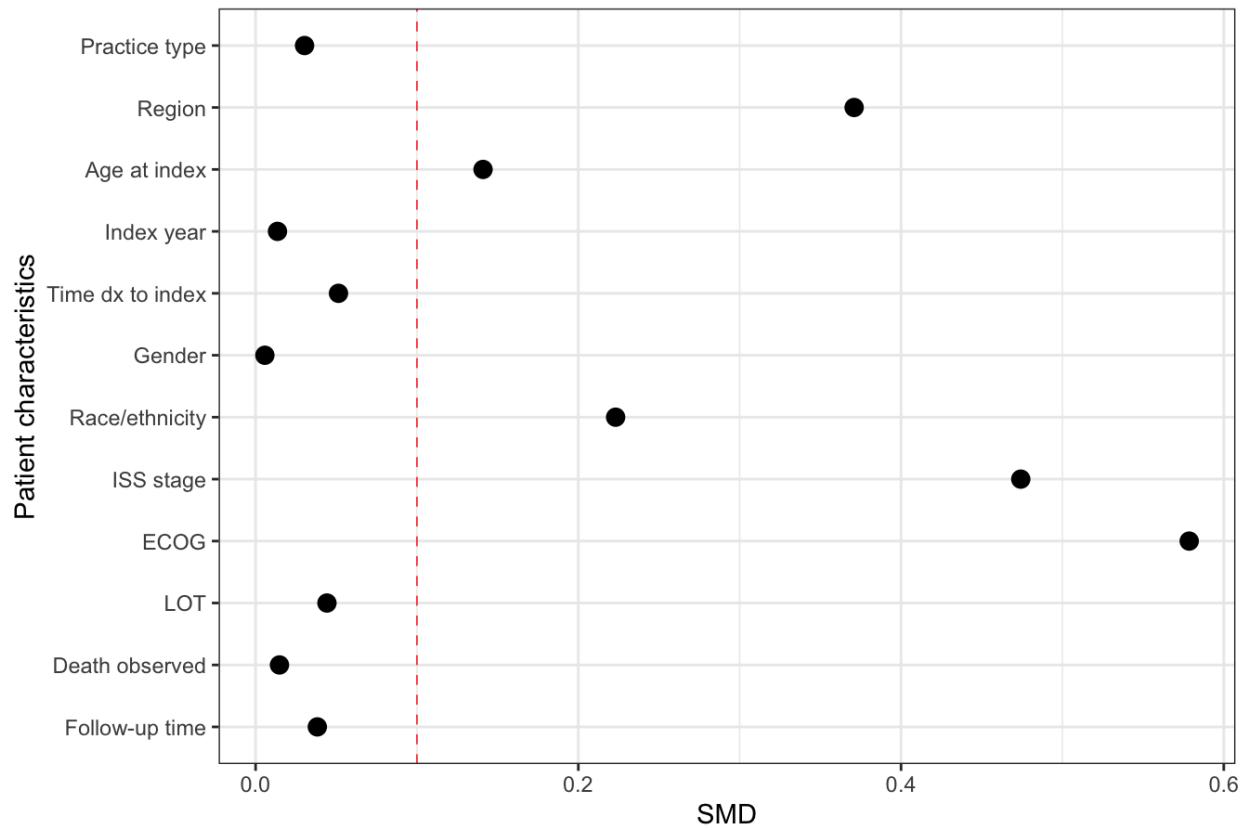
Supplementary Figure 2. Comparison and balance of patient characteristics with and without observed labs in advanced non-small cell lung cancer (top) and multiple myeloma (bottom) real-world cohorts.

Advanced non-small cell lung cancer cohort (aNSCLC)



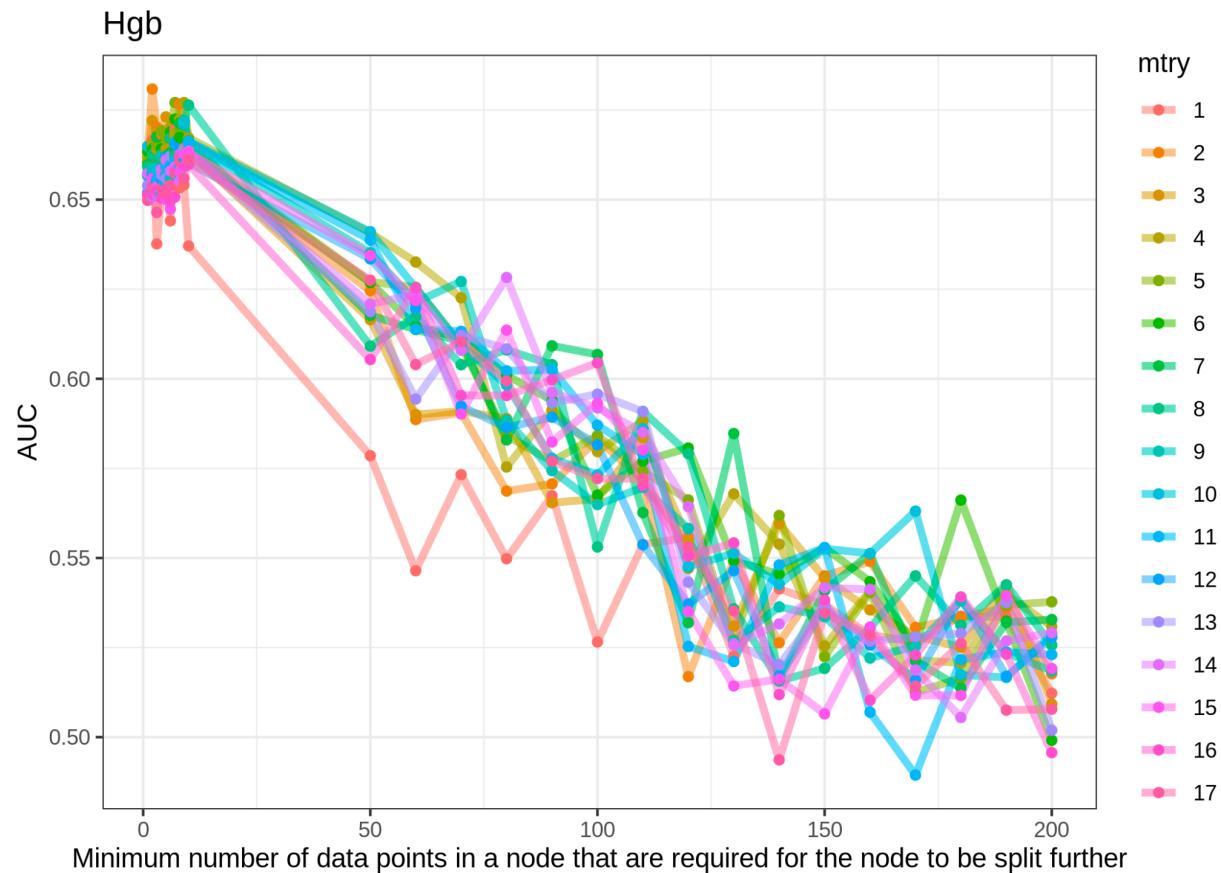
Multiple Myeloma cohort (MM)

Comparing patients with and without observed Hgb lab

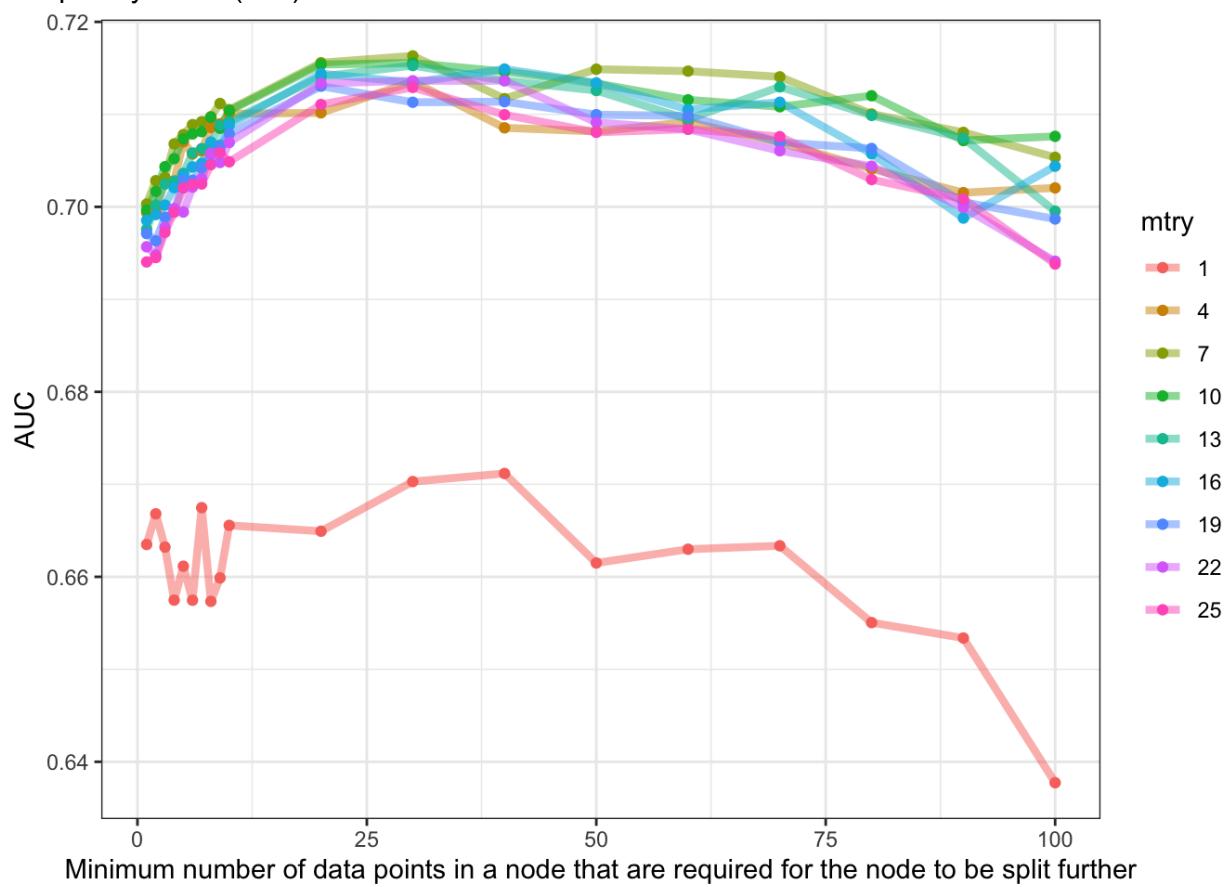


Supplementary Figure 3. Illustration of hyperparameter tuning for AUC diagnostics classifier model in advanced non-small cell lung cancer (top) and multiple myeloma (bottom) real-world cohorts.

Advanced non-small cell lung cancer (aNSCLC)

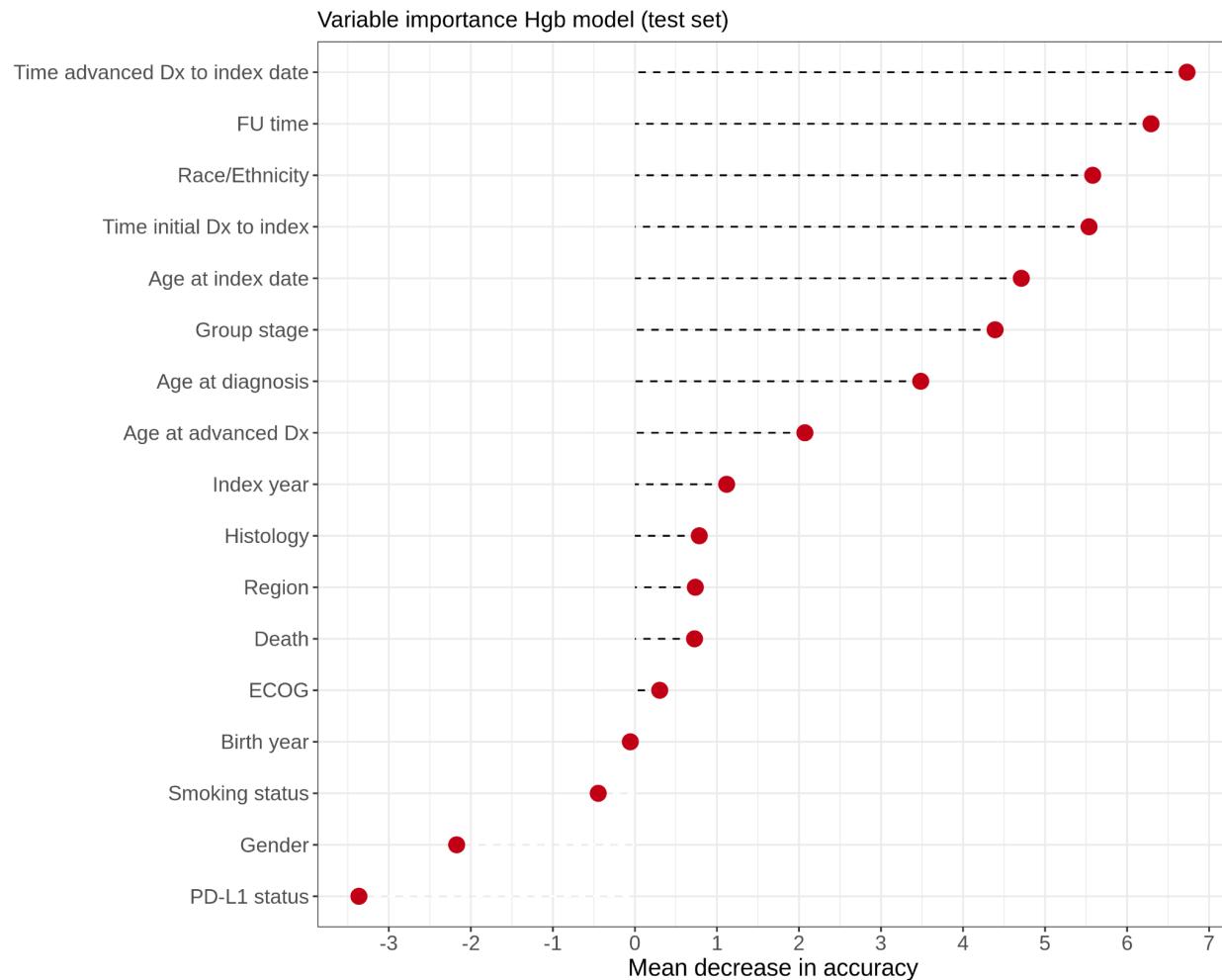


Multiple myeloma (MM)

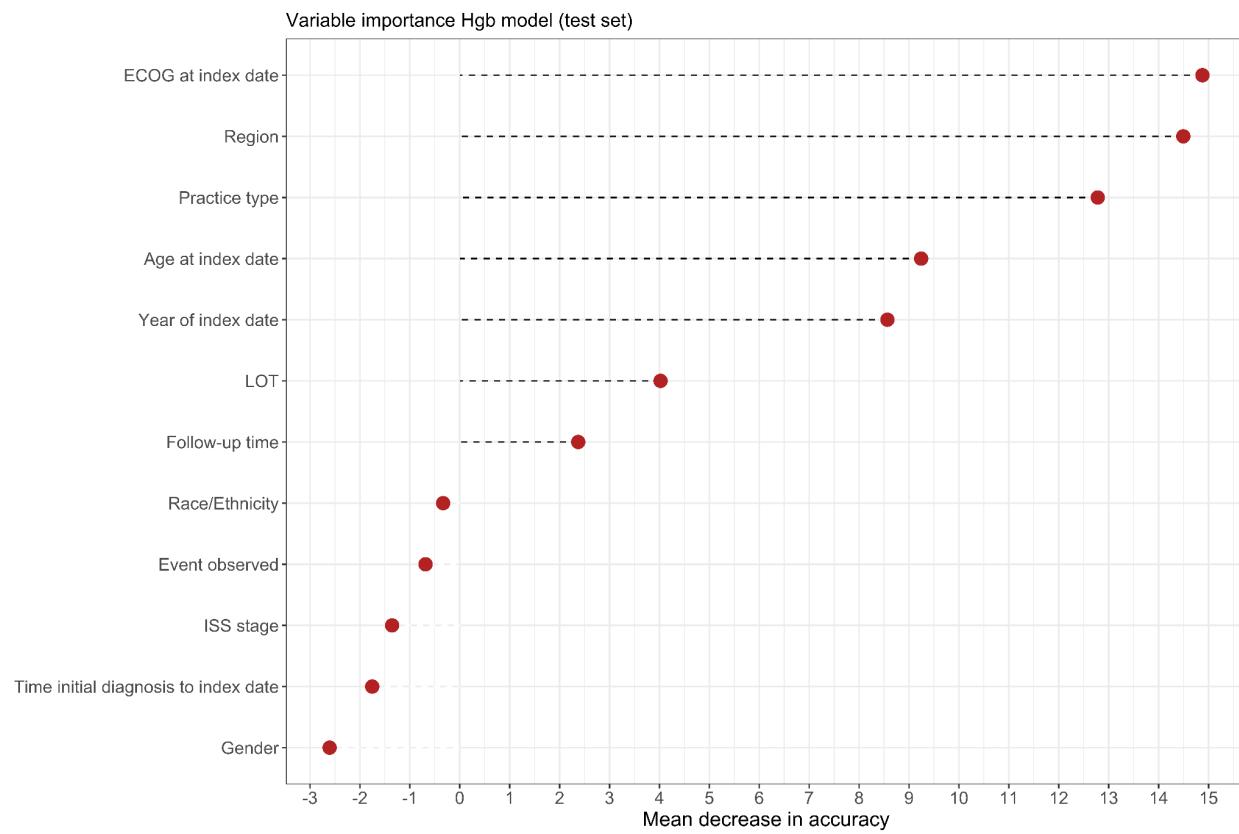


Supplementary Figure 4. Variable importance in predicting missingness in advanced non-small cell lung cancer (top) and multiple myeloma (bottom) real-world cohorts.

Advanced non-small cell lung cancer (aNSCLC)



Multiple Myeloma (MM)



SUPPLEMENTARY TABLES

Supplementary Table 1. Details on applied real-world cohort case studies.

Study design item	Case study #1 - PD-L1 expression among 1L CPI aNSCLC patients	Case study #2 - Comparative effectiveness of VRd vs. KRd among 1L MM patients
Exposure contrast of interest	PD-L1 positive vs. PD-L1 negative	VRd (lenalidomide, bortezomib, dexamethasone) versus KRd (carfilzomib, lenalidomide, dexamethasone)
Index date	Date of 1L initiation	Date of 1L initiation
Outcome of interest	Overall survival	Overall survival
I/E criteria	<ul style="list-style-type: none">• Confirmed aNSCLC diagnosis• Structured activity within 90 days of aNSCLC diagnosis• Exclusion of patients with occult staging• Metastatic 1L setting• Exposure to mono- or combination therapy containing one of pembrolizumab, nivolumab, atezolizumab, durvalumab, cemiplimab• Age \geq 18 years at index date• ECOG < 2• Index date was in 2017 or later	<ul style="list-style-type: none">• Confirmed MM diagnosis• Structured activity within 90 days of MM diagnosis• Received either VRd or KrD as first-line treatment

	<ul style="list-style-type: none"> Known PD-L1 status 	
Expected unbiased result based on published literature	HR < 1.0 indicating survival benefit for PD-L1 positive	Clinical trial OS HR = 0.98 (95% CI, 0.71 to 1.36) for KRd compared to VRd, indicating similar OS in both arms
Modeling of partially observed hemoglobin (Hgb) lab	Indicator variable with "1" indicating Hgb lab is within a normal range and "0" indicating abnormal/outside of normal range	Indicator variable with "1" indicating Hgb lab is within a normal range and "0" indicating abnormal/outside of normal range
% observed missingness in Hgb lab in final analysis cohort	8.3%	18.5%

Table S2. Results of Hotelling's test.

Cohort	Test statistic	p
aNSCLC	146.29	<.0001
MM	465.79	<.0001

Abbreviations: aNSCLC = advanced non-small cell lung cancer, MM = Multiple Myeloma

Covariates compared for aNSCLC: Age at index date, Gender, Index year, Histology, Group stage, Smoking status, Birth year, Race/Ethnicity, Region, ECOG, Age at diagnosis, Age at advanced Dx, Time initial Dx to index, Time advanced Dx to index date, Time from index date to end of follow-up, Censoring indicator, PD-L1 status.

Covariates compared for MM: Age at index date (1L treatment start), ECOG at index date, Gender, ISS stage, Index year (calendar year of 1L treatment start), Practice type, Race/Ethnicity, Region, Time diagnosis to index date, Time from index date to end of follow-up, Censoring indicator, Line of therapy

SUPPLEMENTARY MATERIALS: SIMULATION CODE

sim_functions_v2.R

Code to simulate data from different missingness mechanisms, apply complete case and multiple imputation analyses, and assess bias.

```
library(tidyverse)
library(lubridate)
library(survival)
library(mice)

library(doParallel)
registerDoParallel(cores = detectCores())
print(detectCores())

N <- 5000

simulate_base_data <- function() {

  # Simulate covariate
  Z <- rnorm(N)

  # Simulate lab binary indicator
  lab_prob <- ifelse(Z > 0, 0.1, 0.9)
  lab <- rbinom(N, 1, lab_prob)
  mean(lab)

  # Simulate binary treatment
  p_trt <- plogis(log(0.7) * Z + log(1.3) * lab)
  trt <- rbinom(N, 1, p_trt)
  mean(trt)

  # Linear predictor
  log_hazard <- log(2) * Z + log(0.7) * lab + log(0.8) * trt
```

```

# Simulate OS and censoring
base_data <- data.frame(Z, lab, trt, log_hazard) %>%
  mutate(lambda = 0.3 * exp(log_hazard),
        survival_time = rexp(n(), lambda),
        censoring_time = rexp(n(), lambda),
        time = pmin(survival_time, censoring_time),
        status = as.numeric(censoring_time > survival_time)) %>%
  select(-log_hazard, -lambda, -survival_time, -censoring_time) %>%
  as_tibble()

return(base_data)
}

simulate_base_mnar_data <- function() {

  # Simulate covariate
  Z <- rnorm(N)
  U <- rnorm(N)

  # Simulate lab binary indicator
  lab_prob <- plogis(3 * Z + 3 * U)
  lab <- rbinom(N, 1, lab_prob)
  mean(lab)

  # Simulate binary treatment
  p_trt <- plogis(log(0.7) * Z + log(1.3) * lab)
  trt <- rbinom(N, 1, p_trt)
  mean(trt)

  # Linear predictor
  log_hazard <- log(2) * Z + log(0.7) * lab + log(0.8) * trt

  # Simulate OS and censoring
  base_data <- data.frame(Z, U, lab, trt, log_hazard) %>%
    mutate(lambda = 0.3 * exp(log_hazard),
          survival_time = rexp(n(), lambda),
          censoring_time = rexp(n(), lambda),
          time = pmin(survival_time, censoring_time),
          status = as.numeric(censoring_time > survival_time)) %>%
    select(-log_hazard, -lambda, -survival_time, -censoring_time) %>%
    as_tibble()

  return(base_data)
}

```

```

}

simulate_mcar_data <- function(seed) {

  set.seed(seed)
  base_data <- simulate_base_data()

  # Simulate lab missingness indicator
  sim_data <- base_data %>%
    mutate(p_missing = 0.5,
      lab_missing = rbinom(n(), 1, p_missing),
      lab_obs = ifelse(lab_missing == 0, lab, NA)) %>%
    select(-p_missing, -lab_missing)

  return(sim_data)
}

simulate_mar_data <- function(seed) {

  set.seed(seed)
  base_data <- simulate_base_data()

  # Simulate lab missingness indicator
  sim_data <- base_data %>%
    mutate(p_missing = ifelse(z > 0, 0.1, 0.9),
      lab_missing = rbinom(n(), 1, p_missing),
      lab_obs = ifelse(lab_missing == 0, lab, NA)) %>%
    select(-p_missing, -lab_missing)

  return(sim_data)
}

simulate_mar_hard_data <- function(seed) {

  set.seed(seed)
  base_data <- simulate_base_data()

  # Simulate lab missingness indicator
  sim_data <- base_data %>%
    mutate(p_missing = plogis(1 + 2 * z + log(0.5) * time + log(1.7) * trt),
      lab_missing = rbinom(n(), 1, p_missing),
      lab_obs = ifelse(lab_missing == 0, lab, NA)) %>%

```

```

    select(-p_missing, -lab_missing)

    return(sim_data)
}

simulate_mnar_data <- function(seed) {

  set.seed(seed)
  base_data <- simulate_base_mnar_data()

  # Simulate lab missingness indicator
  sim_data <- base_data %>%
    mutate(p_missing = plogis(1 + 2 * Z + log(0.5) * time + log(1.7) * trt + 2 * U),
      lab_missing = rbinom(n(), 1, p_missing),
      lab_obs = ifelse(lab_missing == 0, lab, NA)) %>%
    select(-U, -p_missing, -lab_missing)

  return(sim_data)
}

simulate_mnar_hard_data <- function(seed) {

  set.seed(seed)
  base_data <- simulate_base_data()

  # Simulate lab missingness indicator
  sim_data <- base_data %>%
    mutate(p_missing = plogis(2 * Z + log(0.5) * time + log(1.7) * trt + 2 * lab),
      lab_missing = rbinom(n(), 1, p_missing),
      lab_obs = ifelse(lab_missing == 0, lab, NA)) %>%
    select(-p_missing, -lab_missing)

  return(sim_data)
}

run_analysis_sims <- function(type,
                                niter) {

  results <- foreach(ix = 1:niter, .combine = "rbind") %dopar% {
    print(ix)
    set.seed(ix)
}

```

```

if (type == "MCAR") {
  sim_data <- simulate_mcar_data(seed = ix)
} else if (type == "MAR") {
  sim_data <- simulate_mar_data(seed = ix)
} else if (type == "MAR-hard") {
  sim_data <- simulate_mar_hard_data(seed = ix)
} else if (type == "MNAR") {
  sim_data <- simulate_mnar_data(seed = ix)
} else if (type == "MNAR-hard") {
  sim_data <- simulate_mnar_hard_data(seed = ix)
}

truth <- coxph(Surv(time, status) ~ trt + Z + lab, data = sim_data) %>%
  broom::tidy(exponentiate = TRUE, conf.int = TRUE) %>%
  transmute(term,
            estimate,
            conf_low = conf.low,
            conf_high = conf.high)

ignore_lab <- coxph(Surv(time, status) ~ trt + Z, data = sim_data) %>%
  broom::tidy(exponentiate = TRUE, conf.int = TRUE) %>%
  transmute(term,
            estimate,
            conf_low = conf.low,
            conf_high = conf.high) %>%
  bind_rows(data.frame(term = "lab",
                        estimate = 0,
                        conf_low = 0,
                        conf_high = 0))

cca <- coxph(Surv(time, status) ~ trt + Z + lab_obs, data = sim_data) %>%
  broom::tidy(exponentiate = TRUE, conf.int = TRUE) %>%
  transmute(term = ifelse(grepl("lab", term), "lab", term),
            estimate,
            conf_low = conf.low,
            conf_high = conf.high)

imp <- mice(sim_data %>%
  mutate(lab_obs = as.factor(lab_obs)) %>%
  select(-lab),
  meth = "logreg", m = 100, maxit = 1, printFlag = FALSE)
fit <- with(imp, coxph(Surv(time, status) ~ trt + Z + lab_obs))
mice <- pool(fit) %>%
  summary() %>%

```

```

as.data.frame() %>%
as_tibble() %>%
transmute(term = ifelse(grepl("lab", term), "lab", as.character(term)),
estimate = exp(estimate),
conf_low = exp(estimate - 1.96*std.error),
conf_high = exp(estimate + 1.96*std.error))

res <- list(truth = truth, cca = cca, mice = mice, ignore_lab = ignore_lab) %>%
bind_rows(.id = "method") %>%
mutate(iter = ix)

res
}

return(results)
}

if (F) {

res_mcarr <- run_analysis_sims("MCAR", 200)
res_mar <- run_analysis_sims("MAR", 200)
res_mar_hard <- run_analysis_sims("MAR-hard", 200)
res_mnar <- run_analysis_sims("MNAR", 200)
res_mnar_hard <- run_analysis_sims("MNAR-hard", 200)

results <- list("MCAR" = res_mcarr,
"MAR" = res_mar,
"MAR-hard" = res_mar_hard,
"MNAR" = res_mnar,
"MNAR-hard" = res_mnar_hard)

save(results, file = "analysis_sim_results.RData")
}

```

tuning_functions.R

Code to fit and tune random forest and logistic regression prediction models to obtain the AUC missingness diagnostic.

```

# Tuning functions
library(tidymodels)
library(tidyverse)
library(pROC)

# Create randomForest model specification to be used in tidymodels framework
# Sets num trees to constant, tunes mtry and min_n
CreateRandomForestSpec <- function(trees = 1000) {

  tune_spec <- rand_forest(
    mtry = tune(),
    trees = trees,
    min_n = tune()) %>%
    set_mode("classification") %>%
    set_engine("randomForest")

  return(tune_spec)
}

# Create logistic regression model specification to be used in tidymodels framework
CreateLogitSpec <- function() {

  tune_spec <- logistic_reg(
    penalty = tune(),
    mixture = 0) %>%
    set_mode("classification") %>%
    set_engine("glmnet")

  return(tune_spec)
}

# Creates prediction workflow given template data and model specification
# - Preprocesses data by dummying all categorical variables (except outcome)
# - Then, prediction model formula is missing indicator ~ all features
# - The prediction model used is that given in model_spec
CreateWorkflow <- function(template_data, model_spec) {

  pred_rec <- recipe(missing_lab ~ ., data = template_data) %>%
    step_dummy(all_nominal(), -all_outcomes())

  tune_wf <- workflow() %>%
    add_recipe(pred_rec) %>%

```

```

add_model(model_spec)

return(tune_wf)
}

# Tunes model specified in workflow over grid_df
# Uses n_cv-fold CV over train_data (default = 5)
TuneModel <- function(train_data, workflow, grid_df, n_cv = 5, par = FALSE) {

  pred_folds <- vfold_cv(train_data, v = n_cv)

  tune_res <- tune_grid(
    object = workflow,
    resamples = pred_folds,
    grid = grid_df,
    control = control_grid(verbose = TRUE, allow_par = par)
  )

  return(tune_res)
}

# Given test set results object, extracts test set ROC curve and metrics computed over varying thresholds
GetTestMetrics <- function(test_res) {

  test_preds <- test_res %>%
    collect_predictions() %>%
    transmute(truth = forcats::fct_relevel(missing_lab, "TRUE", "FALSE"),
              estimate = .pred_TRUE)

  roc <- roc(test_preds$truth, test_preds$estimate, ci = TRUE)

  pos <- test_preds %>% filter(truth == "TRUE") %>% nrow
  neg <- test_preds %>% filter(truth == "FALSE") %>% nrow

  f1_data <- roc_curve(test_preds, truth, estimate) %>%
    mutate(tp = sensitivity * pos,
           tn = specificity * neg,
           fn = pos - tp,
           fp = neg - tn,
           precision = tp / (tp + fp),
           recall = sensitivity,
           f1_score = (2 * precision * recall) / (precision + recall),

```

```

accuracy = (tp + tn) / (tp + tn + fp + fn)

output <- list(roc = roc,
               f1_data = f1_data)

return(output)
}

```

diagnostic_sims.R

Code to simulate datasets subject to missingness, and apply Little's test and the AUC diagnostic.

```

library(tidyverse)
library(lubridate)
library(survival)
library(mice)
library(Hotelling)

source("sim_functions_v2.R")
source("tuning_functions.R")

library(doParallel)
registerDoParallel(cores = detectCores())
print(detectCores())

mar_auc_diagnostic <- function(simulated_data) {

  simulated_data_rf <- simulated_data %>%
    mutate(missing_lab = as.factor(is.na(lab_obs))) %>%
    select(-lab_obs, -lab)

  # Split into training and testing
  simulated_data_split <- initial_split(simulated_data_rf, prop = 0.8)
  simulated_data_train <- training(simulated_data_split)
  simulated_data_test <- testing(simulated_data_split)
}

```

```

# Workflow
rf_spec <- CreateRandomForestSpec()
tune_wf <- CreateWorkflow(simulated_data_train, rf_spec)

# Tuning
rf_grid <- expand.grid(mtry = 1:4,
                        min_n = c(10, 20, 30, 40, 50)) %>%
  as_tibble()
tune_res <- TuneModel(simulated_data_train, tune_wf, rf_grid, n_cv = 5)
best_auc <- select_best(tune_res, "roc_auc")

# Test set fit
final_rf <- finalize_model(rf_spec, best_auc)
final_wf <- CreateWorkflow(simulated_data_train, final_rf)
final_res <- final_wf %>%
  last_fit(simulated_data_split)

# Test set metrics
test_roc <- final_res %>%
  collect_metrics() %>%
  select(.metric, .estimate) %>%
  pivot_wider(names_from = .metric, values_from = .estimate) %>%
  select(roc_auc)

return(as.numeric(test_roc))
}

littles_test <- function(simulated_data) {

  simulated_data_little <- simulated_data %>%
    select(-lab)

  # Recode as matrix for Little's MCAR test
  rw_matrix <- simulated_data_little %>%
    as.matrix

  # Create matrices of observations with missing labs and observations with complete labs
  lab_col_ind <- which(colnames(rw_matrix) == "lab_obs")
  missing_ind <- which(is.na(rw_matrix[, lab_col_ind]))
  complete_ind <- setdiff(1:nrow(rw_matrix), missing_ind)

  matrix_missing <- rw_matrix[missing_ind, -lab_col_ind]
  matrix_complete <- rw_matrix[complete_ind, -lab_col_ind]
}

```

```

# Compare with Hotelling's multivariate t-test
hotelling_out <- hotelling.test(matrix_missing, matrix_complete)
pval <- hotelling_out[[2]]

return(pval)
}

run_diagnostic_sims <- function(type,
                                   niter) {

  results <- foreach(ix = 1:niter, .combine = "rbind") %dopar% {
    print(ix)

    if (type == "MCAR") {
      simulated_data <- simulate_mcar_data(seed = ix)
    } else if (type == "MAR") {
      simulated_data <- simulate_mar_data(seed = ix)
    } else if (type == "MAR-hard") {
      simulated_data <- simulate_mar_hard_data(seed = ix)
    } else if (type == "MNAR") {
      simulated_data <- simulate_mnar_data(seed = ix)
    } else if (type == "MNAR-hard") {
      simulated_data <- simulate_mnar_hard_data(seed = ix)
    }

    mcar_pval <- littles_test(simulated_data)
    mar_auc <- mar_auc_diagnostic(simulated_data)

    c(mcar_pval, mar_auc, ix)
  }

  results <- as.data.frame(results)
  colnames(results) <- c("mcar_pval", "mar_auc", "iter")
  rownames(results) <- NULL
  results <- as_tibble(results)

  return(results)
}

if (T) {

```

```
res_mcar <- run_diagnostic_sims(type = "MCAR", niter = 200)
res_mar <- run_diagnostic_sims(type = "MAR", niter = 200)
res_mar_hard <- run_diagnostic_sims(type = "MAR-hard", niter = 200)
res_mnar <- run_diagnostic_sims(type = "MNAR", niter = 200)
res_mnar_hard <- run_diagnostic_sims(type = "MNAR-hard", niter = 200)

results <- bind_rows(res_mcar %>% mutate(mechanism = "MCAR"),
                      res_mar %>% mutate(mechanism = "MAR"),
                      res_mar_hard %>% mutate(mechanism = "MAR-hard"),
                      res_mnar %>% mutate(mechanism = "MNAR-easy"),
                      res_mnar_hard %>% mutate(mechanism = "MNAR-hard"))

save(results, file = "diagnostic_sim_results.RData")
}
```